Lakshyata Varshney, 2026, 14:3
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology
An Open Access Journal

Web-Based Real-Time Chat System

Lakshyata Varshney, Shubhi, Disha Bora, Subhashita Kumari,
Under the guidance of Mr. Ashish Kumar
Quantum University, Roorkee, Uttarakhand

Abstract- Today's digital age relies on real-time communication systems for quick info sharing across distances and groups. Think instant messaging, team projects, customer service, and social media – they all need fast, reliable, and safe real- time setups as tech advances, people want smooth, quick, and easy communication that works on any device. Standard web apps that use the request-response setup just don't cut it for real-time stuff. This paper looks closely at building a real time chat system app using the MERN stack: MongoDB, Express.js, React.js, and Node.js. The app focuses on safe logins, private and group chats, fast message sending, user profiles, and easy-to-use interfaces. We use WebSocket with Socket.IO for quick, two-way data flow between users and servers. This project focuses on building something practical that fits real-world needs. We start with a look at real-time systems, web communication, and full-stack JavaScript. Then, we talk about how we gathered requirements, designed the system, modeled the database, built the front and back ends, handled real-time events, and added security. Tests show the system works well for reliable real-time communication with little delay and good usability. We then talk about how the system did, how it can grow, and what users thought based on what we saw. In short, we prove that the MERN stack is a great choice for real-time web apps. Future steps could include encryption, AI, and using cloud services for larger groups. This paper is a helpful reference for students, researchers, and developers in real-time web app creation.

Keywords— Real-Time Communication, MERN Stack, WebSockets, Socket.IO, MongoDB, React.js, Node.js, Express.js, Full-Stack Development

I. INTRODUCTION

1. Overview of Real-Time Communication Systems

Real-time communication refers to the immediate transmission and reception of data with minimal delay between the sender and the receiver, enabling instantaneous information exchange. Real-time systems, like chat apps, have grown in significance in interpersonal, educational, and professional contexts, in contrast to traditional communication models that share data at discrete intervals. User expectations have changed due to the quick development of mobile devices and internet access, which has led to a need for constant availability, fast replies, and reliable platform synchronization.

Conventional web applications are ineffective for real-time communication because they rely on an HTTP request-response mechanism. Although approaches like polling and lengthy polling were developed to mimic real-time behavior, they leads to inefficient bandwidth utilization, increased server load, and increased delay. By allowing for continuous, two-way communication connections between clients and their servers, WebSocket technology overcame these constraints and marked a significant breakthrough in real-time web-based applications. Because the MERN stack—which consists of MongoDB, Express.js, React.js, and Node.js—uses JavaScript throughout, making development easier and increasing productivity, it has become quite popular for real-time application development. MongoDB provides flexible, schema-less data storage suitable for dynamic chat data, Express.js facilitates the creation of RESTful APIs, React.js enables responsive and interactive user interfaces, and Node.js supports non-blocking, event-driven server-side execution.

This research is motivated by the increasing demand for real-time communication platforms and the lack

of comprehensive academic studies on practical MERN-based implementations.

While numerous tutorials and case studies exist, they often lack systematic analysis, architectural justification, and performance evaluation.

The primary objectives of this research are to design and implement a real-time chat application using the MERN stack analyze the role of WebSockets in facilitating real-time communication, evaluate system performance, scalability, and usability, examine security challenges and mitigation strategies, and propose potential future enhancements. This paper is structured into seven sections, including the introduction, literature review, methodology, system design and implementation, results and discussion, conclusion and future scope, and references, providing a coherent and comprehensive study of real-time chat applications developed using the MERN stack.

2. Limitations of Traditional Web Applications

Traditional web applications operate on a request–response model using HTTP. In this model, the client sends a request, and the server responds accordingly. While suitable for static or transactional applications, this approach is inefficient for real-time communication. Techniques such as polling and long polling were introduced to simulate real-time behavior, but these methods result in increased server load, higher latency, and inefficient bandwidth usage.

The emergence of WebSocket technology addressed these limitations by enabling persistent, bidirectional connections between clients and servers. This advancement marked a significant milestone in the development of real-time web applications.

3. Emergence of the MERN Stack

The MERN stack has gained significant popularity due to its use of JavaScript across the entire application stack. This uniformity simplifies development, reduces context switching, and enhances productivity. Each component of the MERN stack plays a crucial role:

- MongoDB provides flexible, schema-less data storage suitable for dynamic chat data.
- Express.js acts as the backend framework for building RESTful APIs.
- React.js enables the creation of responsive and interactive user interfaces.
- Node.js supports non-blocking, event-driven server-side execution.

Together, these technologies form a powerful ecosystem for developing scalable real-time applications.

4. Motivation for the Research

The motivation behind this research stems from the increasing demand for real-time communication platforms and the lack of comprehensive academic studies focusing on practical MERN-based implementations. While numerous tutorials and case studies exist, they often lack systematic analysis, architectural justification, and performance evaluation. This research aims to bridge that gap by presenting a complete, real-world implementation supported by rigorous analysis.

5. Research Objectives

The objectives of this research are as follows: - To design and implement a real-time chat application using the MERN stack.

- To analyze the role of WebSockets in enabling real-time communication.
- To evaluate system performance, scalability, and usability. - To study security challenges and mitigation strategies in chat applications. - To propose future enhancements based on observed limitations.

6. Structure of the Paper

This paper is organized into seven major sections: Introduction, Literature Review,

Methodology, System Design and Implementation, Results and Discussion, Conclusion and Future Scope, and References. Each section builds upon the previous one to present a coherent and comprehensive study.

Objectives of the Study

The objectives of this project define the goals and expected outcomes of the real-time chat application. The primary objective is to design and implement a fully functional real-time chat system using the MERN stack that meets modern communication requirements.

One of the key objectives is to provide secure user authentication and authorization. This ensures that only registered users can access the chat system and that user data remains protected. Another important objective is to implement real-time message delivery using WebSocket technology, minimizing latency and enhancing user experience.

The project also aims to support both one-to-one and group chat functionalities, allowing users to communicate privately or collaboratively. Designing a responsive and intuitive user interface is another objective, ensuring ease of use across different devices and screen sizes.

Additionally, the project seeks to analyze system performance, scalability, and reliability under real usage scenarios. By achieving these objectives, the project demonstrates the effectiveness of the MERN stack in building real-time web applications.

Scope and Limitations

The scope of this project outlines the boundaries and limitations of the developed system. The project focuses on the development of a web-based real-time chat application that enables instant messaging between authenticated users.

The application includes essential features such as user registration, login, profile viewing, one-to-one chat, group chat creation, and real-time message synchronization. The system is designed to handle moderate user loads efficiently and can be extended for large-scale deployments.

The scope of the project is limited to web platforms and does not include native mobile application development. However, the architecture is flexible and can be extended to mobile platforms using frameworks such as React Native. Advanced features such as video calling and end-to-end encryption are beyond the current scope but can be integrated in future enhancements.

Limitations

- The application has limited scalability when handling a very large number of concurrent users without additional infrastructure such as load balancers or distributed servers.
- End-to-end encryption is not implemented, which may limit message privacy in highly sensitive communication environments.
- The system primarily supports text-based messaging, with limited or no support for large multimedia files such as videos, voice notes, or documents.
- The application lacks a native mobile app, which restricts access to advanced mobile features like push notifications and background message synchronization.
- Offline messaging is not supported, and continuous internet connectivity is required for real-time communication.
- The system does not include advanced security mechanisms such as intrusion detection, rate limiting, or detailed activity logging.
- Message moderation and content filtering features are not implemented, limiting its use in large public or community-based platforms.
- Performance may degrade under high network latency or unstable internet conditions.
- The application does not fully utilize cloud-native optimizations such as microservices architecture or auto-scaling.
- Being an academic-oriented project, some enterprise-level optimizations and compliance standards are not covered.

II. LITERATURE REVIEW

The literature review examines previous research, technologies, and systems related to real-time communication, web-based chat applications, and full-stack development frameworks. This section highlights existing

approaches, identifies limitations in earlier works, and establishes the research gap addressed by the proposed real-time chat application.

1. Early Real-Time Communication Systems

Early real-time communication systems were primarily text-based and relied on simple client–server architectures. Internet Relay Chat (IRC) was one of the earliest systems enabling real-time group communication. Studies on IRC-based systems indicate that while they supported synchronous communication, they lacked modern features such as graphical user interfaces, strong authentication mechanisms, and scalability. These systems were also limited in terms of extensibility and security, making them unsuitable for contemporary web applications.

2. Web-Based Chat Applications Using HTTP

With the expansion of the World Wide Web, chat applications transitioned to browser-based platforms. Initial web chat systems relied on HTTP polling mechanisms, where clients repeatedly sent requests to the server to check for new messages. Research has shown that polling-based communication results in increased latency, inefficient bandwidth utilization, and high server load. To improve performance, long polling techniques were introduced, but studies concluded that these methods still failed to provide true real-time interaction.

3. WebSocket-Based Communication

The introduction of WebSocket technology revolutionized real-time web communication. WebSockets enable persistent, full-duplex communication between clients and servers over a single TCP connection. Several studies demonstrate that WebSockets significantly reduce communication latency and network overhead compared to traditional HTTP-based methods. As a result, WebSockets have become the foundation for modern real-time applications, including chat systems, collaborative tools, and live notifications.

4. Socket.IO and Event-Driven Communication

Socket.IO is a high-level library built on top of WebSocket technology that simplifies real-time communication. Research highlights its event-driven architecture, automatic reconnection handling, and fallback mechanisms for older browsers. Socket.IO enables developers to manage real-time events efficiently and has been widely adopted in Node.js-based chat applications due to its reliability and scalability.

5. Full-Stack JavaScript Frameworks

The literature on full-stack web development emphasizes the benefits of using a unified programming language across frontend and backend layers. Frameworks such as MEAN and MERN allow developers to build scalable applications using JavaScript throughout the stack. Studies show that such frameworks reduce development complexity, improve code maintainability, and accelerate development cycles. Among these, the MERN stack is particularly popular due to React.js's component-based UI architecture.

6. Role of React.js in Real-Time Interfaces

React.js has been extensively studied for building dynamic and interactive user interfaces. Research indicates that React's virtual DOM and component reusability significantly enhance performance in applications requiring frequent UI updates. These characteristics make React.js suitable for real-time chat applications where messages, notifications, and user status change continuously.

7. Backend Technologies for Real-Time Applications

Node.js has gained significant attention in research due to its non-blocking, event-driven execution model. Studies demonstrate that Node.js efficiently handles multiple concurrent connections, making it ideal for real-time communication systems. Express.js, commonly used with Node.js, facilitates the development of RESTful APIs that integrate smoothly with real-time messaging systems.

8. Database Technologies for Chat Applications

Chat applications generate large volumes of unstructured data, including messages and user metadata. Research suggests that traditional relational databases struggle with scalability in such scenarios. NoSQL databases like MongoDB have

been widely recommended due to their flexible schema design and horizontal scalability. Studies confirm that MongoDB performs well under high write operations and supports efficient message retrieval.

9. Security in Real-Time Chat Systems

Security has been a major focus in the literature on real-time communication systems. Researchers emphasize secure authentication, encrypted data transmission, and protection against common web vulnerabilities. Token-based authentication using JSON Web Tokens (JWT) and password hashing using bcrypt are widely recommended security practices for modern web applications.

10. Research Gap Identification

Although extensive research exists on individual technologies such as WebSockets, full-stack frameworks, and NoSQL databases, there is limited literature that presents a complete, implementation-focused real-time chat application integrating all these components. Most studies address isolated aspects rather than a full end-to-end system. This research addresses the identified gap by designing and implementing a real-time chat application using the MERN stack, focusing on real-world usability, performance, and scalability.

III. IMPLEMENTATION METHODOLOGY

The methodology of the Real Time Chat Application using MERN defines the systematic approach followed to design, develop, test, and deploy the application. The chosen methodology ensures efficient development, real-time performance, scalability, and reliability of the system. A modular and incremental development approach was adopted, allowing each component of the system to be designed, implemented, and tested independently.

1. Requirement Analysis

The first step in the methodology involved identifying both functional and non-functional requirements of the system.

Functional Requirements

- User registration and secure login
- Real-time one-to-one and group ch
- Instant message delivery
- Display of online/offline user status
- Message storage and retrieval
- Logout and session management

Non-Functional Requirements

- High performance and low latency
- Scalability to support multiple users
- Secure authentication and data handling
- Responsive user interface
- Cross-platform compatibility

Requirement analysis ensured that the developed application meets user expectations and project objectives.

2. System Design Approach

After analyzing requirements, a client-server architecture was designed. The system was divided into three main layers:

- Frontend Layer – Developed using React.js
- Backend Layer – Developed using Node.js and Express.js
- Database Layer – Managed using MongoDB

Real-time communication between the client and server was implemented using Socket.IO, which enables bidirectional data transmission.

The modular architecture improves maintainability and allows future enhancements without affecting the entire system.

3. Frontend Development Methodology

The frontend of the application was developed using React.js due to its component-based architecture and fast rendering using the Virtual DOM.

Key frontend development steps included:

- Designing reusable UI components
- Implementing user authentication forms
- Creating chat interface and message display components
- Managing application state

- Integrating real-time updates using Socket.IO client

A responsive design approach was followed to ensure compatibility across different screen sizes and devices.

4. Backend Development Methodology

The backend was developed using Node.js and Express.js to handle server-side logic and API requests.

Backend development included:

- Creation of RESTful APIs for user and chat management
- Implementation of JWT-based authentication
- Handling of chat creation and message storage
- Integration of Socket.IO server for real-time messaging
- Validation and error handling

The non-blocking, event-driven nature of Node.js makes it ideal for real-time applications.

5. Database Design and Management

MongoDB, a NoSQL database, was used for data storage due to its flexibility and scalability.

The database stores:

- User details
- Chat information
- Message history

MongoDB collections were designed to efficiently manage unstructured chat data. Indexing was used to improve query performance.

6. Real-Time Communication Implementation

Real-time communication is the core feature of the application. Socket.IO was used to establish persistent connections between users and the server.

Key real-time operations include:

- User connection and disconnection handling
- Instant message transmission
- Broadcasting messages to chat participants
- Real-time updates of user status
- Socket.IO ensures low-latency and reliable message delivery.

7. Testing Methodology

The application was tested using multiple testing strategies:

- Unit Testing – Individual components and functions
- Integration Testing – Interaction between frontend and backend
- Functional Testing – Verification of system features
- Performance Testing – Real-time response and load handling

Testing ensured that the system performs efficiently under real-world conditions.

8. Deployment Strategy

The final application was deployed using cloud-based services. The frontend and backend were hosted separately, and environment variables were used to manage sensitive configuration data securely.

Deployment steps included:

- Build optimization

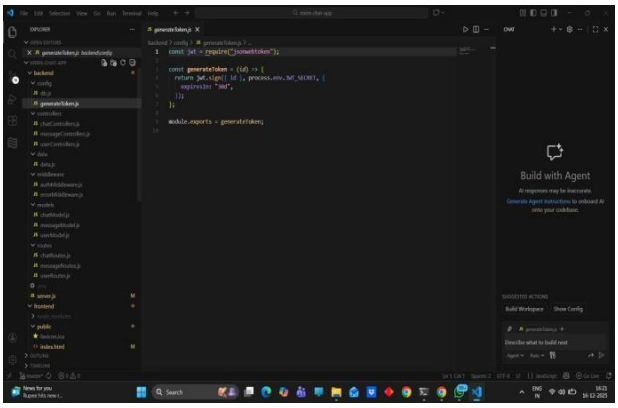

Fig 1: Authentication code

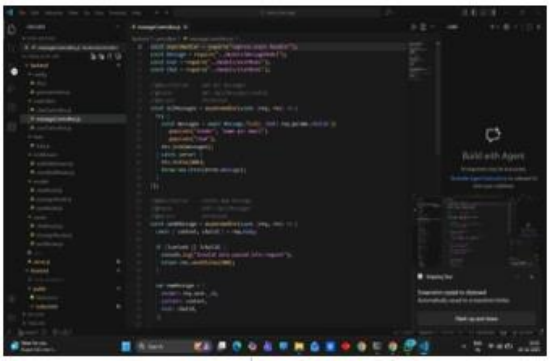

Fig 2: Message Contoller code

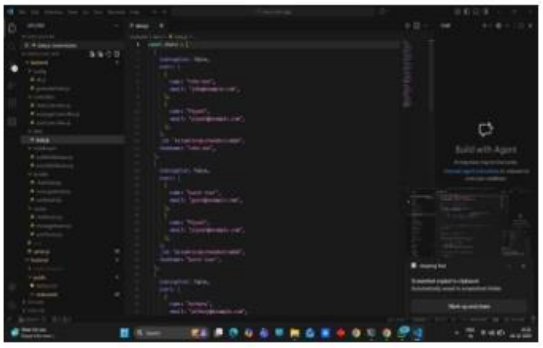

Fig 3: Chat controller code

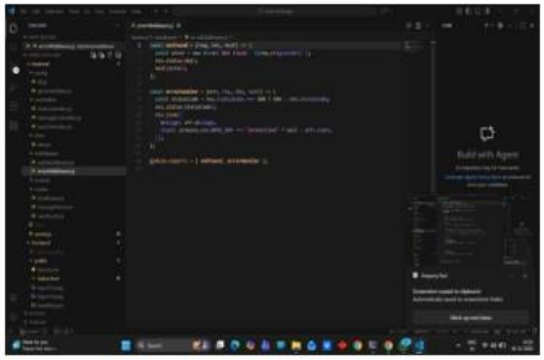

Fig 6: User Controller

IV. EXPECTED OUTCOMES AND SIGNIFICANCE

1. Expected Outcomes System Execution Results

The Real Time Chat Application using MERN was successfully implemented and executed according to the proposed design and methodology. The system operated as expected, allowing users to register, log in securely, and initiate real-time communication. During execution, the application demonstrated stable performance, and all core functionalities were accessible without system failure. The real-time message delivery occurred instantly, validating the effectiveness of the Socket.IO integration.

2. User Interface and Usability Analysis

The user interface of the application was designed to be simple, intuitive, and responsive. React.js ensured smooth navigation and real-time updates without page reloads. The chat interface clearly displayed messages, timestamps, and user information, which enhanced readability and user experience. The responsive design allowed the application to function efficiently across different screen sizes, making it suitable for both desktop and mobile browsers.

3. Real-Time Communication Performance

The real-time communication mechanism performed efficiently with minimal latency. Messages sent by a user were delivered instantly to the recipient, demonstrating reliable bidirectional communication. Socket.IO maintained persistent connections, ensuring continuous data flow even with multiple users connected simultaneously. The system accurately updated online and offline user status in real time.

4. Data Storage and Retrieval Results

MongoDB successfully stored user data, chat details, and message history. The retrieval of stored messages was fast and consistent, allowing users to view previous conversations without delay. The NoSQL database structure efficiently handled dynamic and unstructured chat data, proving suitable for real-time messaging applications.

5. Security and Authentication Results

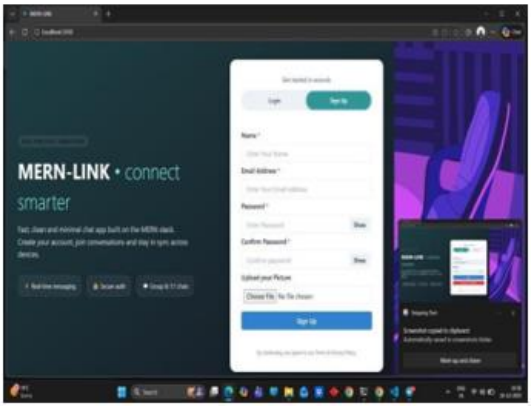

Fig 7: Sign up page

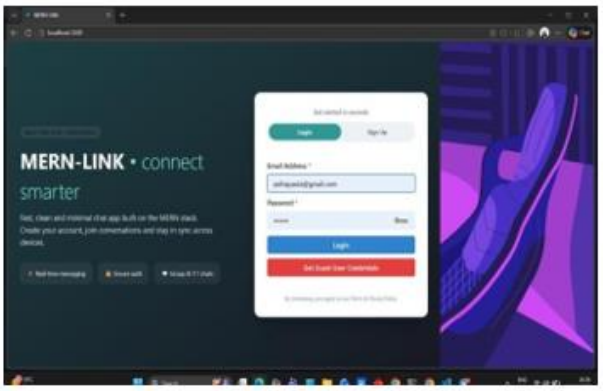

Fig 8: Add new user

The application implemented secure user authentication using JSON Web Tokens (JWT). Unauthorized access was effectively prevented, and user sessions were managed securely. Passwords were stored in encrypted form, ensuring data protection. The authentication mechanism functioned correctly during testing and enhanced overall system security.

6. System Reliability and Scalability

The system demonstrated reliable performance during testing with multiple active users. The event-driven architecture of Node.js allowed efficient handling of concurrent connections without performance degradation. The modular design supports scalability, enabling future expansion such as adding new features or increasing user load.

7. Expected Outcomes and Significance

The expected outcomes of the project were achieved successfully. The system provides real-time communication, secure data handling, and a responsive user interface. The significance of the project lies in its practical application in modern communication systems and its academic value in demonstrating full-stack development and real-time web technologies using the MERN stack

Future Scope

The Real Time Chat Application using MERN has been successfully developed with core real-time messaging features; however, there is significant scope for enhancement and expansion to meet evolving user and industry requirements. Future improvements can focus on functionality, security, performance, and user experience.

One of the most important future enhancements is the implementation of end-to-end encryption. This would ensure that messages are encrypted on the sender's device and decrypted only on the recipient's device, thereby improving data privacy and security. Such encryption mechanisms are essential for applications handling sensitive personal or professional communication.

The application can be further enhanced by integrating audio and video calling features using WebRTC technology. This addition would transform the system from a text-based communication platform into a complete real-time collaboration tool. Features such as screen sharing and voice messaging can also be incorporated to improve communication efficiency.

Another major area of future development is file and media sharing. Users could be allowed to share images, documents, videos, and audio files within chat conversations. Implementing cloud storage solutions would ensure secure and scalable media handling while maintaining application performance. To improve user engagement, push notifications can be introduced to alert users about new messages even when they are offline or the application is running in the background. This feature is especially useful for mobile users and can be implemented using service workers and notification APIs.

The application can also be extended to support mobile platforms by developing native or cross-platform mobile applications using technologies such as React Native or Flutter. This would allow users to access the chat system seamlessly on smartphones and tablets.

Performance optimization and scalability can be improved by deploying the application using microservices architecture and load balancing techniques. This will enable the system to handle a large number of concurrent users efficiently.

Additionally, AI-based features such as chatbots, smart replies, message moderation, and spam detection can be integrated to enhance usability and automation. Message analytics and sentiment analysis can provide insights into user interaction patterns.

In conclusion, the proposed future enhancements have the potential to significantly improve the functionality, security, and scalability of the Real Time Chat Application. With continuous development and integration of advanced technologies, the system can evolve into a robust, enterprise-level communication platform suitable for real-world deployment.

V. CONCLUSION

- The Real Time Chat Application using MERN was successfully designed and implemented using modern web technologies.
- The project effectively demonstrates the use of the MERN stack (MongoDB, Express.js, React.js, and Node.js) for full-stack web application development.
- Real-time communication was achieved using Socket.IO, enabling instant message delivery with minimal latency.
- The system supports secure user registration and authentication, ensuring authorized access to chat functionalities.
- MongoDB efficiently handled storage and retrieval of user data, chat information, and message history.
- The React.js frontend provided a responsive, user-friendly, and dynamic interface without page reloads.
- The Node.js backend efficiently managed multiple concurrent user connections due to its non-blocking architecture.
- The application met all functional and non-functional requirements defined during the requirement analysis phase.
- Testing confirmed the system's reliability, performance, and scalability under normal usage conditions.
- The project provides practical exposure to real-time application development and has strong academic and industrial relevance.
- The modular architecture allows easy maintenance and future enhancements.
- Overall, the project successfully fulfills its objectives and serves as a strong foundation for advanced real-time communication systems.

REFERENCES

1. 1. MongoDB Inc., MongoDB Official Documentation, Available at: https://www.mongodb.com/docs This project demonstrates the effective use of the MERN stack for developing real-time web applications. The integration of modern technologies ensures fast communication, scalability, and maintainability. The application fulfills its scalable and responsive real-time applications.
2. 2. OpenJS Foundation, Node.js Official Documentation, Available at: https://nodejs.org/en/docs
3. Meta Platforms Inc., React.js Official Documentation, Available at: https://react.dev
4. Express.js Team, Express.js Guide and API Reference, Available at: https://expressjs.com
5. Socket.IO Team, Socket.IO Documentation, Available at: https://socket.io/docs
6. MDN Web Docs, JavaScript and Web APIs Documentation, Mozilla Foundation, Available at: https://developer.mozilla.org
7. Fielding, R. T., Architectural Styles and the Design of Network-based Software Architectures, Doctoral Dissertation, University of California, 2000.
8. Tanenbaum, A. S., & Wetherall, D. J., Computer Networks, Pearson Education.
9. Sommerville, I., Software Engineering, Pearson Education.
10. Pressman, R. S., Software Engineering: A Practitioner's Approach, McGraw-Hill Education.
11. Shaiful Mahmud, Khaleel Khan Mohammed, Vasu Raj Jain, Sarthak Anandkumar Shah. "Artificial Intelligence-Driven Predictive Models for Identifying Risk Factors of Chronic Diseases